Numerical Optimization for Machine Learning

Global Optimizaiton, Subgradients, and Cutting Planes

Mark Schmidt

University of British Columbia

Summer 2022 - Summer 2023

Last Time: Projected Gradient

- We discussed the projected gradient method for constrained optimization,

$$w^{k+1} = \text{proj}_{\mathcal{C}}[w^k - \alpha_k \nabla f(w^k)],$$

  where "proj" returns the closest point inside the constraint set $\mathcal{C}$.
  - Equivalent to minimizing a quadratic approximation of f over $\mathcal{C}$.
  - Non-expansiveness, gradient mapping, projection theorem.
  - Has similar convergence properties to unconstrained gradient descent.
  - Many simple sets $\mathcal{C}$ allow efficient projection.
  - Identifies the active constraints in a finite number of iterations.
- We discussed faster-converging accelerated and Newton-like variants.
  - Acceleration is straightforward, Newton is not straightforward.
- We discussed variants with cheaper iterations.
  - Projected coordinate descent for bound constraints.
  - Projected stochastic gradient for optimizing sums.
  - Frank-Wolfe when linear optimization over set is easier.

Complexity of Minimizing Real-Valued Functions

- Consider minimizing real-valued functions over the unit hyper-cube,

$$\min_{w \in [0,1]^d} f(w),$$

  where f may be non-convex.
- You can use any algorithm you want.

  (simulated annealing, gradient descent + random restarts, genetic algorithms, Bayesian optimization,...)
- How many zero-order oracle calls t before we can guarantee $f(w^t) - f(w^*) \leq \epsilon$?
  - In the worst case: unbounded!

- Given any algorithm, we can construct an f where $f(w^k) - f(w^*) > \epsilon$ forever.
  - Due to real numbers being uncountable.

    (See: `https://mathwithbaddrawings.com/2016/11/09/pick-a-truly-random-number`)
  - Make $f(w) = 0$ except at w^* where $f(w^*) = -2\epsilon$.

    (the w^* is algorithm-specific)

- To say anything in oracle model we need assumptions on f.

Complexity of Minimizing Lipschitz-Continuous Functions

- One of the simplest assumptions is that f is Lipschitz-continuous,

$$|f(w) - f(v)| \leq L\|w - v\|.$$

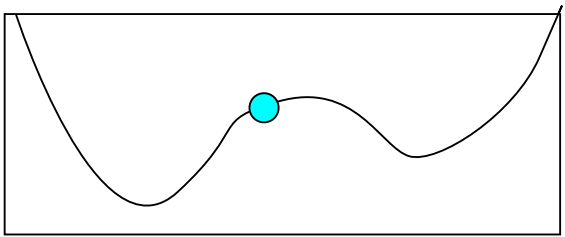
- Function cannot change arbitrarily fast as you change x.

Complexity of Minimizing Lipschitz-Continuous Functions

- One of the simplest assumptions is that f is Lipschitz-continuous,

$$|f(w) - f(v)| \leq L\|w - v\|.$$

- Function cannot change arbitrarily fast as you change x.

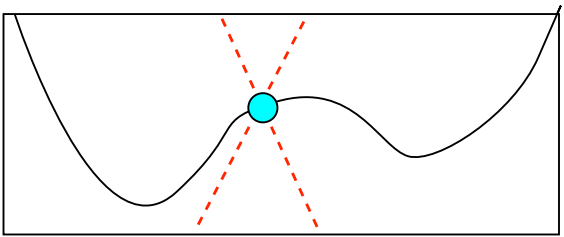


Complexity of Minimizing Lipschitz-Continuous Functions

- One of the simplest assumptions is that f is Lipschitz-continuous,

$$|f(w) - f(v)| \leq L\|w - v\|.$$

- Function cannot change arbitrarily fast as you change x.

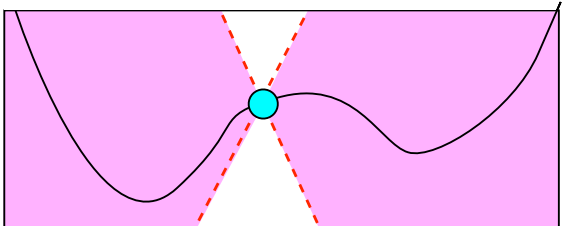


Complexity of Minimizing Lipschitz-Continuous Functions

- One of the simplest assumptions is that f is Lipschitz-continuous,

$$|f(w) - f(v)| \leq L\|w - v\|.$$

- Function cannot change arbitrarily fast as you change x.

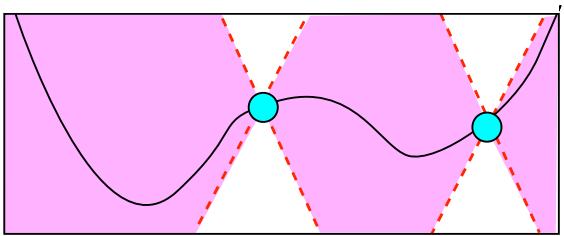


Complexity of Minimizing Lipschitz-Continuous Functions

- One of the simplest assumptions is that f is Lipschitz-continuous,

$$|f(w) - f(v)| \leq L\|w - v\|.$$

- Function cannot change arbitrarily fast as you change x.



Digression: Lipschitz-Continuos vs. Lipschitz-Smooth

- Function f is Lipschitz-cont. if $|f(w) - f(v)| \leq L\|w - v\|$ for some L.
- Gradient ∇f is Lipschitz-cont. if $\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|$ for some L.
  - This is the assumption we used for gradient descent.
  - We say f is"Lipschitz smooth" here.

- Note that neither implies the other:
  - $f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^i w^\top x^i))$ is Lipschitz-cont. with ∇f Lipschitz-cont.
  - $f(w) = \|w\|_1$ is Lipschitz-cont. but does not have a Lipschitz-cont. gradient.
  - $f(w) = \frac{1}{2}\|Xw - y\|^2$ is not Lipshitz-cont. but does have a Lipschitz-cont. gradient.
  - $f(w) = w^4$ is neither Lipschitz-cont nor does it have a Lipschitz-cont gradient.

Complexity of Minimizing Lipschitz-Continuous Functions

- Consider minimizing real-valued functions over the unit hyper-cube,

$$\min_{w \in [0,1]^d} f(w),$$

  where we assume f is Lipschitz continuous (but may be non-convex).
  - This implies f is bounded below.

- With only this assumption, any algorithm requires at least $\Omega(1/\epsilon^d)$ iterations.
  - In zero-order and first-order oracle model.

- An optimal $O(1/\epsilon^d)$ worst-case rate is achieved by a grid-based search method.
  - Evaluating the function at evenly-spaced values of w.

Faster Algorithms for Global Optimization?

- You can also achieve optimal rate in expectation by random guesses.
  - Lipschitz-continuity implies there is a ball of ϵ-optimal solutions around w^*.
  - The radius of the ball is $\Omega(\epsilon)$ so its area is $\Omega(\epsilon^d)$.
  - If we succeed with probability $\Omega(\epsilon^d)$, we expect to need $O(1/\epsilon^d)$ trials.

(mean of geometric random variable)

- Many more-complicated global optimization algorithms exist.
  - Simulated annealing, genetic algorithms, Bayesian optimization, and so on.

- But none of these beat random guessing in the worst case (in oracle model).
  - Which is surprising and a bit depressing.

- Of course, we can solve problems more quickly under stronger assumptions.
  - If f is Lipschitz and convex, subgradient methods only require $O(1/\epsilon^2)$ iterations.

Harmless Global Optimization

- Some global optimization methods are worse than random in the worst case.
  - They can get stuck exploring areas far from the global optimum for too long.
  - I call these harmful optimizers.

- A harmless global optimization algorithm is one that achieves the $O(1/\epsilon^d)$ rate.
  - So using the method is not worse than random guessing.

- How to make any algorithm harmless:
  - On every second iteration, try a random guess.
    - Or try a random guess at any fixed interval or with any fixed probability.

- Other sensible harmless variations:
  - Every t iterations, try the w that is furthest away from all previous w^k.
  - If you know L, you can use differences in previous $f(w^k)$ to prune space (bonus).

Bayesian Optimization (BO)

- Popular approach for hyper-parameter optimization is Bayesian optimization:
  - Build a regression model to predict $f(w)$ based on previous w^k and $f(w^k)$ values.
    - Typically Gaussian processes, but could use random forests (SMAC) or neural nets.
  - Optimize an acquisition function over all of w to choose the next iterate.
    - Expected(improvement), probability(improvement), entropy search, UCB.

- Not faster than random in worst case for optimizing Lipschitz functions.
- Some implementations are not harmless (sometimes worse than random).
  - This is particularly due to the optimizer itself having hyper-parameters.
  - But as we discussed, it is easy to make them harmless.

- For suitably smooth functions a variant of BO has convergence rate of $O(1/\epsilon^{d/\nu})$.
  - Where ν is a measure of smoothness that can be larger or smaller than 1.

Outline

1 Complexity of Global Optimization

2 Subgradients and Subgradient Method

3 Smooth Approximations of Non-Smooth Functions

4 Faster Subgradient Methods

Complexity of Minimizing Convex Functions

- Many optimization problem arising in machine learning are non-smooth,

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2 + \lambda \sum_{j=1}^{d}|w_j|, \quad f(w) = \sum_{i=1}^{n}\max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2}\sum_{j=1}^{d} w_j^2,$$

  including L1-regularized least squares and SVMs.

- We cannot compute $\nabla f(w)$ for all w for non-smooth functions.
  - Absolute value $|w_j|$ has no gradient whenever $w_j = 0$.
  - Hinge loss $\max\{0, 1 - y_i w^T x_i\}$ has no gradient whenever $1 - y_i w^T x_i = 0$.

- Nevertheless, we can compute subgradients of these functions.

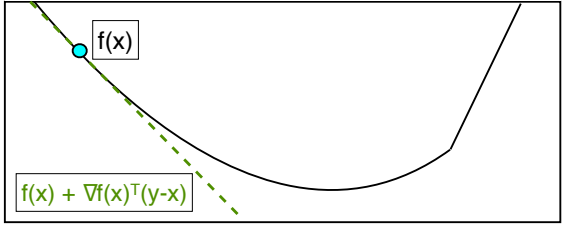
Sub-Gradients and Sub-Differentials

Differentiable convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^\top (v - w), \forall w, v.$$

A vector d is a subgradient of a convex function f at w if

$$f(v) \geq f(w) + d^\top (v - w), \forall v.$$



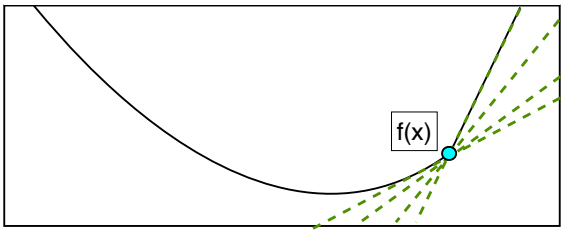
Sub-Gradients and Sub-Differentials

Differentiable convex functions are always above tangent,

$$f(v) \geq f(w) + \nabla f(w)^\top (v - w), \forall w, v.$$

A vector d is a subgradient of a convex function f at w if

$$f(v) \geq f(w) + d^\top (v - w), \forall v.$$



Sub-Gradients and Sub-Differentials Properties

- We can have a set of subgradients called the sub-differential, $\partial f(w)$.
  - Subdifferential is all the possible "tangent" lines.

- For convex functions:
  - Sub-differential is always non-empty (except some weird degenerate cases).
    - Formally, sub-differential guaranteed non-empty on "relative interior".
  - At differentiable w, the only subgradient is the gradient: $\partial f(w) = \{\nabla f(w)\}$.
  - At non-differentiable w, there will be a convex set of subgradients.

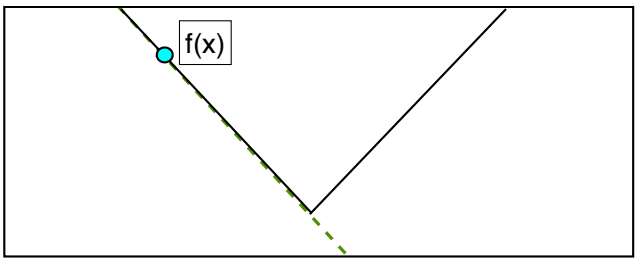
  - We have $0 \in \partial f(w)$ iff w is a global minimum.
    - This generalizes the condition that $\nabla f(w) = 0$ for differentiable functions.

Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial |w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

  - "Sign of the variable if it is non-zero, anything in $[-1, 1]$ if it's zero."

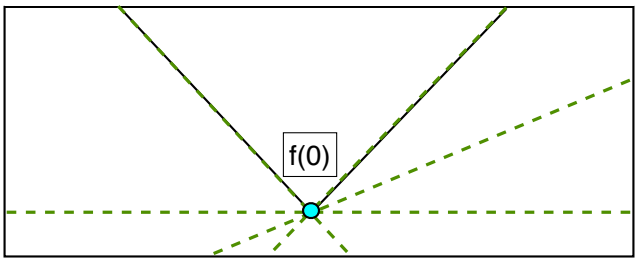


Example: Sub-Differential of Absolute Function

- Sub-differential of absolute value function:

$$\partial |w| = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \\ [-1, 1] & w = 0 \end{cases}$$

  - "Sign of the variable if it is non-zero, anything in $[-1, 1]$ if it's zero."



Sub-Differential of Common Operations

- Some convenient rules for calculating subgradients of convex functions:
  - Sub-differential of differentiable functions only contains gradient:

$$\partial f(w) \equiv \{\nabla f(w)\}.$$

  - Sub-differential of sum is all sum of subgradients of individual functions:

$$\partial(f_1(x) + f_2(x)) = d_1 + d_2 \quad \text{for any} \quad d_1 \in \partial f_1(x), d_2 \in \partial f_2(x).$$

  - Sub-differential of non-negative scaling is scaling of sub-differential,

$$\partial \alpha f(w) = \alpha \partial f(w),$$

    for $\alpha > 0$.
  - Sub-differential of composition with affine function works like the chain rule:

$$\partial f_1(Aw) = A^\top \partial f_1(z), \quad \text{where} \quad z = Aw,$$

Sub-Differential of Common Operations

- Some convenient rules for calculating subgradients of convex functions:
  - Sub-differential of max is all convex combinations of argmax gradients:

$$\partial \max\{f_1(w), f_2(w)\} = \begin{cases} \nabla f_1(w) & f_1(x) > f_2(w) \\ \nabla f_2(w) & f_2(x) > f_1(w) \\ \underbrace{\theta \nabla f_1(w) + (1-\theta)\nabla f_2(w)}_{\text{for all } 0 \leq \theta \leq 1} & f_1(w) = f_2(w) \end{cases}$$

- Max rule gives sub-differential of absolute value, using that $|\alpha| = \max\{\alpha, -\alpha\}$.
- Max rule also gives simple way to get sub-differential of ReLU or hinge loss,

$$\partial \max\{0, 1 - y_i w^T x_i\} \equiv \begin{cases} 0 & 1 - y_i w^T x_i > 0 \\ -y_i x_i & 1 - y_i w^T x_i < 0 \\ \underbrace{\theta 0 + (1-\theta)(-y_i x_i)}_{\text{for all } 0 \leq \theta \leq 1} & 1 - y_i w^T x_i = 0 \end{cases}.$$

Why does L1-Regularization but not L2-Regularization give Sparsity?

- Consider L2-regularized least squares,

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

- Element j of the gradient at $w_j = 0$ is given by

$$\nabla_j f(w) = x_j^\top \underbrace{(Xw - y)}_{r} + \lambda 0.$$

- For $w_j = 0$ to be a solution, we need $0 = \nabla_j f(w^*)$ or that

$$0 = x_j^\top r^* \quad \text{where } r^* = Xw^* - y \text{ for the solution } w^*$$

that column j is orthogonal to the final residual.

  - This is possible, but it is very unlikely (probability 0 for random data).
  - Increasing λ doesn't help.

Why does L1-Regularization but not L2-Regularization give Sparsity?

- Consider L1-regularized least squares,

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|_1.$$

- Element j of the subdifferential at $w_j = 0$ is given by

$$\partial_j f(w) \equiv x_j^\top \underbrace{(Xw - y)}_{r} + \lambda \underbrace{[-1, 1]}_{\partial|w_j|}.$$

- For $w_j = 0$ to be a solution, we need $0 \in \partial_j f(w^*)$ or that

$$0 \in x_j^T r^* + \lambda[-1, 1] \quad \text{or equivalently}$$
$$-x_j^T r^* \in \lambda[-1, 1] \quad \text{or equivalently}$$
$$|x_j^\top r^*| \leq \lambda,$$

that column j is "close to" orthogonal to the final residual.

- So features j that have little to do with y will often lead to $w_j = 0$.
- Increasing λ makes this more likely to happen.

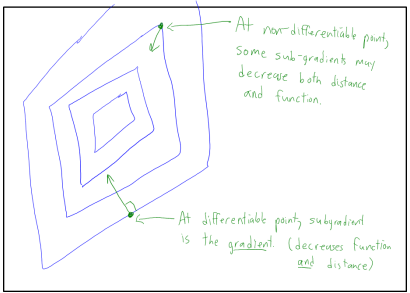
Subgradient Method

- The basic subgradient method:

$$w^{k+1} = w^k - \alpha_k g_k,$$

for any $g_k \in \partial f(w^k)$.

- This can increase the objective even for small α_k.
  - Though for convex f the distance to solutions decreases:
    - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough α_k (and sub-optimal w^k).



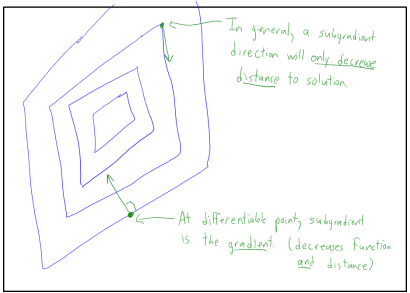
Subgradient Method

- The basic subgradient method:

$$w^{k+1} = w^k - \alpha_k g_k,$$

for any $g_k \in \partial f(w^k)$.

- This can increase the objective even for small α_k.
  - Though for convex f the distance to solutions decreases:
    - $\|w^{k+1} - w^*\| < \|w^k - w^*\|$ for small enough α_k (and sub-optimal w^k).



Convergence Rate of Subgradient for Lipschitz+Convex Functions

- Proofs that analyze $\|w^k - w^*\|$ usually start with the following steps:

$$\begin{aligned}
\|w^{k+1} - w^*\|^2 &= \|(w^k - \alpha_k g_k) - w^*\|^2 && \text{(definition of } w^{k+1}\text{)} \\
&= \|(w^k - w^*) - \alpha_k g_k\|^2 && \text{(group terms)} \\
&= \|w^k - w^*\|^2 - 2\alpha_k g_k^T(w^k - w^*) + \alpha_k^2\|g_k\|^2 && \text{(expand squared norm)}
\end{aligned}$$

- The terms on the right are similar to the terms we get in descent lemma.
  - Typically use a lower bound on size of second term and upper bound on third term.
- If we assume f is Lipschitz and convex, and g_k is a subgradient then we have

$$\begin{aligned}
&\|g_k\|^2 \leq L^2 && \text{(Lipshitz implies subgradients are bounded by } L\text{)} \\
&f(w^*) \geq f(w^k) + g_k^T(w^* - w^k) && \text{(using } w^k \text{ and } w^* \text{ in definition of subgradient)}
\end{aligned}$$

and the second property can be re-written as $-g_k^T(w^k - w^*) \leq -[f(w^k) - f(w^*)]$.

Convergence Rate of Subgradient for Lipschitz+Convex Functions

- For the subgradient method applied to Lipschitz and convex f we thus have

$$\begin{aligned}\|w^{k+1} - w^*\|^2 &= \|w^k - w^*\|^2 - 2\alpha_k \underbrace{g_k^T(w^k - w^*)}_{\geq f(w^k) - f(w^*)} + \alpha_k^2 \underbrace{\|g_k\|^2}_{\leq L^2} \\ &\leq \|w^k - w^*\|^2 - 2\alpha_k[f(w^k) - f(w^*)] + \alpha_k^2 L^2.\end{aligned}$$

- Re-arranging we get

$$2\alpha_k[f(w^k) - f(w^*)] \leq \|w^k - w^*\|^2 - \|w^{k+1} - w^*\|^2 + \alpha_k^2 L^2,$$

  and summing/telescoping over values of $k = 1$ to t we get

$$2\sum_{k=1}^{t} \alpha_k[f(w^k) - f(w^*)] \leq \|w^0 - w^*\|^2 - \|w^{k+1} - w^*\|^2 + L^2 \sum_{k=1}^{t} \alpha_k.$$

- Using f^b as the lowest $f(w^k)$ and $\|w^{k+1} - w^*\|^2 \geq 0$ we can re-arrange to get...

Convergence Rate of Subgradient for Lipschitz+Convex Functions

- A bound that is very similar to what we showed for SGD:

$$f(w^b) - f(w^*) \leq \frac{\|w^1 - w^*\|^2 + L^2 \sum_{k=1}^t \alpha_k^2}{2 \sum_{k=1}^t \alpha_k}$$
$$= O\left(\frac{1}{\sum_k \alpha_k}\right) + O\left(\frac{\sum_k \alpha_k^2}{\sum_k \alpha_k}\right).$$

- We get the same conclusions as for SGD based on the step size choices:
  - Using decreasing $\alpha_k = \alpha/k$ gives a rate of $O(1/\log(k))$.
    - In low dimensions, this is worse than random guessing.
  - Using decreasing $\alpha_k = \alpha/\sqrt{k}$ gives a rate of $O(\log(k)/\sqrt{k})$.
    - Optimal convergence rate up to a log factor.
  - Using constant $\alpha_k = \alpha$ gives a rate of $O(1/\alpha k) + O(\alpha)$.
    - Faster convergence up to an accuracy depending on the constant α.

Polyak Step Size for Subgradient Method

- Backtracking may not work for the subgradient method.
  - There may be no step size α_k that decreases the objective f.
  - And we cannot backtrack based on $\|w^k - w^*\|$ since we do not know w^*.

- We can improve from $O(\log(k)/\sqrt{k})$ to $O(1/\sqrt{k})$ using the Polyak step size,

$$\alpha_k = \frac{f(w^k) - f^*}{\|g_k\|^2},$$

  which requires on a lower bound f^* on $f(w^*)$ and minimizes the upper bound $\|w^{k+1} - w^*\|^2 \leq \|w^k - w^*\|^2 - 2\alpha_k[f(w^k) - f^*] + \alpha_k^2\|g_k\|^2$.
  - This can increase or decrease the step size between iterations.
  - This makes α_k go to zero as we approach the optimum (w^* becomes a fixed point).
  - This leads to the mentioned $O(1/\epsilon^2)$ iteration complexity.

Convergence Rate of Projected Gradient

- The projected subgradient method for optimizing over a convex set $\mathcal{C}$,

$$w^{k+1} = \text{proj}_{\mathcal{C}}[w^k - \alpha_k g_k],$$

for any $g_k \in \partial f(w^k)$.

  - For example, our earlier problem of optimizing over the unit hyper-cube.

- To analyze this method, we can use non-expansiveness of projection::

$$\begin{aligned} \|w^{k+1} - w^*\|^2 &= \|\text{proj}_{\mathcal{C}}[w^k - \alpha_k g_k] - \text{proj}_{\mathcal{C}}[w^*]\|^2 \quad \text{(definition of } w^{k+1}\text{, } w^* \text{ feasibility)} \\ &\leq \|(w^k - \alpha_k g_k) - w^*\|^2 \quad \text{(non-expansiveness)}, \end{aligned}$$

and then the remaining steps are the same as the unconstrained case.

  - We get the same rates as the unconstrained case.

Step Sizes based on Diameter Bound

- If $\mathcal{C}$ is bounded and we have an upper bound D on $\|w - w^*\|^2$ over all w:
  - We get a rate of $O(1/\sqrt{k})$ using adaptive $\alpha_k = \frac{\sqrt{2D}}{\|g_k\|\sqrt{k}}$.
    - Similar to Polyak step size, this achieves the $O(1/\epsilon^2)$ rate without the log factor.
- If we also know the Lipschitz constant L:
  - We get a rate of $O(1/\sqrt{t})$ after exactly t iterations using constant $\alpha_k = \frac{DL}{\sqrt{t}}$.
    - A constant step size depending on the number of iterations we use.

Convergence Rate of Stochastic Subgradient

- The stochastic subgradient method uses

$$w^{k+1} = w^k - \alpha_k g_{i_k},$$

  where g_{i_k} is a subgradient for a random training example.

- This method has the same convergence rate as deterministic subgradient method.
  - Upper bound holds with some additional expectations appearing.

- For the SVM problem:
  - Deterministic subgradient needs $O(1/\epsilon^2)$ iterations and n subgradients per iteration.
  - Stochastic subgradients needs $O(1/\epsilon^2)$ iterations and 1 subgradient per iteration.
    - So do not use the deterministic subgradient method for finite sum problems.
- Also note that SGD and stochastic subgradient have the same convergence rate.
  - So SGD is not slowed down when the objective is non-smooth.

Subgradient Methods for Strongly-Convex Objectives

- For strongly-convex objectives:
  - Convergence of subgradient and stochastic subgradient is $O(\log(k)/k)$.
    - Using a step size of $\alpha_k = 1/\mu k$, based on the average iterate $\bar{w}_k = \frac{1}{k}\sum_{t=1}^{k} w^k$.
  - Can improve this to $O(1/k)$ using averages that bias towards later iterates.
    - Suffix averaging computes average over second half of the iterations.
    - Can use $\alpha_k = 2/\mu(k+1)$ and average proportional to $k+1$.
- However, for most problems I do not recommend using the above step size.
  - Can move exponentially-fast away from optimum before slowly moving towards it.
    - Usually $\mu = O(1/n)$ or $O(1/\sqrt{n})$ initial step size is something like $O(n)$.
  - Works well for binary SVMs, but often does not work well in practice.
  - Requires knowing μ, and rate degrades substantially if over-estimated.
    - Not "robust" to the step size.
  - For ML, often better to use a constant step size (or constant then later decrease).
    - Get a robust $O(1/k)$ convergence to a neighbour of solution.

Subgradient Methods for Strongly-Convex Objectives

- Another weird thing about when f is Lipschitz and strongly-convex:
  - These functions cannot exist over all of $\mathbb{R}^d$.
    - Lipschitz-continuity means subgradients are bounded.
    - Strong-convexity requires subgradients to be unbounded.
  - However, these functions can exist over a bounded set $\mathcal{C}$.

- Thus, for strongly-convex we typically discuss projected stochastic subgradient.
  - Where we project onto a bounded set.

Subgradient Methods for Non-Convex Objectives

- For non-convex functions, "global" subgradients may not exist for every w.
- We can define subgradients "locally" around current w (Clarke subdifferential).
  - This is how you define "gradient" of ReLU function in neural networks.

- Subgradient method not known to converge for general non-convex f.

- Many not-too-strong assumptions exist under which it converges:
  - Weakly-convex functions are functions f where $f(w) + \frac{\mu}{2}\|w\|^2$ is convex for some μ.
    - Includes all convex functions and all Lipschitz-smooth functions (may be non-convex)..
    - Includes composition of convex and Lipschitz-smooth (may be non-smooth).
    - Can show $O(1/\sqrt{k})$ convergence rate in this setting, same as Lipschitz-smooth.
  - Tame functions are a very-general class where stochastic subgradient converges.
    - Includes neural networks with ReLU activations.

Outline

1 Complexity of Global Optimization

2 Subgradients and Subgradient Method

3 Smooth Approximations of Non-Smooth Functions

4 Faster Subgradient Methods

Smooth Approximations to Non-Smooth Functions

- In CPSC 340, we used smooth approximations to non-smooth functions.
  - Absolute value can be approximated by Huber's function,

$$|w| \approx \begin{cases} \frac{1}{2}w^2 & |w| \leq \delta, \\ \delta(|w| - \frac{1}{2}\delta) & |w| > \delta. \end{cases}$$

    - So you would approximate linear regression under the L1-norm as

$$f(w) = \sum_{i=1}^{n} |w^T x_i - y_i| \approx \sum_{i=1}^{n} h_\delta(w^T x_i - y_i).$$

  - Maximum can be approximated by log-sum-exp,

$$\max_j(w_j) \approx \log(\sum_j \exp(\delta w_j)),$$

    for a temperature parameter δ.
  - Exist other approximations like Huberized SVM (better behaved than log-sum-exp).
    - Or the GeLU activation used to smooth ReLU in GPT3.

To Smooth or Not to Smooth?

- Key advantage of smoothing: you can use deterministic methods.
  - Get faster convergence rates than subgradient method.
    - $O(1/\sqrt{\epsilon})$ for subgradient vs $O(1/\epsilon^2)$ for accelerated gradient for convex.
    - $O(\log(1/\epsilon))$ for subgradient vs $O(1/\epsilon)$ for accelerated gradient for strongly-convex.
  - Can use line search to set step size.
  - Can use momentum/acceleration/Newton-like methods for faster convergence.

- Reasons you may not want to smooth:
  - Smoothing can destroy the structure in the solution.
    - L1-regularization yields sparsity due to the non-smoothness, Huber would remove this.
  - In some cases the smooth approximation is expensive.
    - Arises in structured SVMs (log-sum-exp approximation may be NP-hard to compute).
  - Smooth methods do not converge faster in stochastic setting

.

- Accurate smooth approximations may be hard to optimize...

Does Smoothing Help?

- Exist smoothing methods for generic smoothing of many non-smooth convex f
  - Like "write f in terms of its Fenchel conjugate with a strongly-convex regularizer".
  - Or "optimize the Moreau envelope" (adds L2-regularization to objective evaluation).

- For a given ϵ, allows us to construct a differentiable function f_ϵ where

$$f(w) \leq f_\epsilon(w) \leq f(w) + \epsilon,$$

  so that minimizing $f_\epsilon(w)$ gets us within ϵ of the optimal solution.

- Problem: the Lipschitz constant $L = O(1/\epsilon)$.
  - Implies gradient descent is not faster than subgradient method.
  - But we can get faster rates by applying accelerated methods to smooth problem.

Does Smoothing Help?

- Consider differentiable f_ϵ that is within ϵ of f and has $L = O(1/\epsilon)$.

- If f is convex and we apply gradient descent then we need

$$t = \underbrace{O(L/\epsilon)}_{\text{smoothed problem}} = \underbrace{O(1/\epsilon^2)}_{\text{original problem}},$$

  the same iteration complexity as the subgradient method.
  - No gain from smoothing.

- If instead we apply accelerated gradient descent then we need

$$t = O(\sqrt{L/\epsilon}) = O(1/\epsilon),$$

  which is faster than subgradient methods.
  - Converges at the speed of unaccelerated gradient descent.

Does Smoothing Help?

- Consider differentiable f_ϵ that is within ϵ of f and has $L = O(1/\epsilon)$.

- For strongly-convex functions if we apply gradient descent then we need

$$t = O(L \log(1/\epsilon)) = O((1/\epsilon) \log(1/\epsilon)),$$

  which is actually worse than the subgradient methods by a log factor.
  - But the log factor can be removed using a sequence of f_ϵ with decreasing ϵ.

- If instead we apply accelerated gradient descent then we need

$$t = O(\sqrt{L} \log(1/\epsilon)) = O((1/\sqrt{\epsilon}) \log(1/\epsilon)),$$

  which is faster than subgradient methods (but not linear convergence).
  - Get the rate of accelerated methods for non-strongly convex functions.

Does Smoothing Help?

- Consider finite sum differentiable f_ϵ that is within ϵ of f and has $L = O(1/\epsilon)$.

- For strongly-convex functions if we apply SAG/SVRG then we need

$$t = O((n + L)\log(1/\epsilon)) = O((n + 1/\epsilon)\log(1/\epsilon)),$$

  which is worse than the stochastic subgradient method (due to the n and log).

- If instead we apply accelerated variance-reduced methods then we need

$$t = O((n + \sqrt{nL})\log(1/\epsilon)) = O((n + \sqrt{n/\epsilon})\log(1/\epsilon)),$$

  which can be faster than subgradient methods.
  - But not linear convergence (and again, annealing can remove the $\log(1/\epsilon)$ factor).

To Smooth or Not to Smooth?

- In practice, smoothing will probably help a lot.
  - Because you can do line-search, accelerated/Newton-like methods, and so on.
  - And because the gradient is always a descent direction.

- Further, you may not care that smoothed problem is ϵ-close to original.
  - Huberized versions often have same test error, even with non-small ϵ.
  - In this setting, why worry about solving the original problem?
    - Just solve the smoothed problem quickly.

- Cases where you do not want to smooth:
  - Smooth approximation is much more expensive to evaluate.
    - For structured SVM over matching polytope, poly-time vs. NP-hard.
  - You are using stochastic subgradient with a small batch size.
    - For batch size of 3 million in GPT3, smoothing the ReLUs probably makes sense.
  - You have a particular non-smooth structure in the solution (like L1-regularization).
    - Though subgradient methods are not particularly good with this issue either.

Outline

1 Complexity of Global Optimization

2 Subgradients and Subgradient Method

3 Smooth Approximations of Non-Smooth Functions

4 Faster Subgradient Methods

Faster Subgradient Methods?

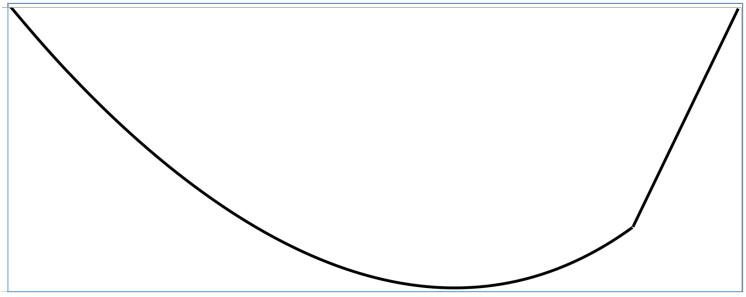
- For smooth optimization, we discussed accelerated gradient methods.
  - Improve iteration complexity for convex and strongly-convex problems.

- Can we develop accelerated subgradient methods?
  - No! (At least in terms of dimension-independent rates.)
  - Subgradient methods are optimal in a first-order oracle model.
    - Where at each iteration we receive $f(w^k)$ and a $g_k \in \partial f(w^k)$.
  - Thus, no faster method is possible in the worst case.

- However, there are faster dimension-dependent subgradient-based algorithms.
  - We also have faster subgradient-based method in practice for many applications.

Bisection: Linear Convergence in 1 Dimension

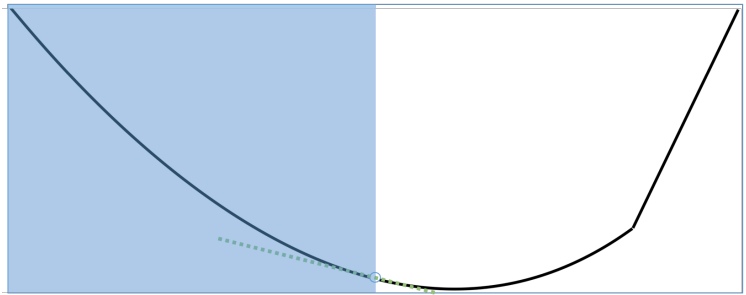
- Consider minimizing a $d = 1$ convex function over an interval:



- Consider the following bisection method for finding a minimizer:
  - At each iteration, compute a subgradient at the middle of the interval.
  - Set the lower/upper bound of the interval to the midpoint (using subgradient sign).
- At each step, maximum distance to a minimizer w^* is cut in half.
  - Achieves linear convergence with rate 0.5, giving iteration complexity of $O(\log(1/\epsilon))$.

Bisection: Linear Convergence in 1 Dimension

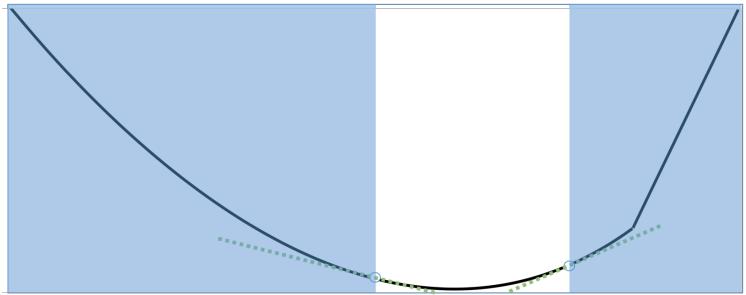
- Consider minimizing a $d = 1$ convex function over an interval:



- Consider the following bisection method for finding a minimizer:
  - At each iteration, compute a subgradient at the middle of the interval.
  - Set the lower/upper bound of the interval to the midpoint (using subgradient sign).
- At each step, maximum distance to a minimizer w^* is cut in half.
  - Achieves linear convergence with rate 0.5, giving iteration complexity of $O(\log(1/\epsilon))$.

Bisection: Linear Convergence in 1 Dimension

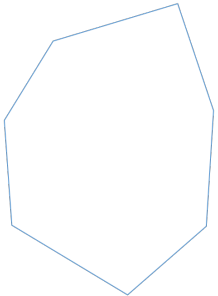
- Consider minimizing a $d = 1$ convex function over an interval:



- Consider the following bisection method for finding a minimizer:
  - At each iteration, compute a subgradient at the middle of the interval.
  - Set the lower/upper bound of the interval to the midpoint (using subgradient sign).
- At each step, maximum distance to a minimizer w^* is cut in half.
  - Achieves linear convergence with rate 0.5, giving iteration complexity of $O(\log(1/\epsilon))$.

Cutting Plane: Linear Convergence in d Dimensions

- Cutting plane methods generalize bisection to higher dimensions.
  - For minimizing convex functions over bounded polyhedrons.



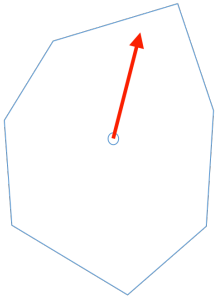
- At each iteration, compute a subgradient at the "center" of the polyhedral.
  - From the definition of subgradient we have for any w that

$$f(w) \geq f(w^k) + g_k^T(w - w^k),$$

    so any w satisfying $g_k^T(w - w^k) > 0$ will be greater than $f(w^k)$.
  - Adding this constraint corresponds to a plane that "cuts" the polyhedron.

Cutting Plane: Linear Convergence in d Dimensions

- Cutting plane methods generalize bisection to higher dimensions.
  - For minimizing convex functions over bounded polyhedrons.



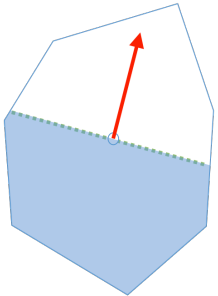
- At each iteration, compute a subgradient at the "center" of the polyhedral.
  - From the definition of subgradient we have for any w that

$$f(w) \geq f(w^k) + g_k^T(w - w^k),$$

    so any w satisfying $g_k^T(w - w^k) > 0$ will be greater than $f(w^k)$.
  - Adding this constraint corresponds to a plane that "cuts" the polyhedron.

Cutting Plane: Linear Convergence in d Dimensions

- Cutting plane methods generalize bisection to higher dimensions.
  - For minimizing convex functions over bounded polyhedrons.



- At each iteration, compute a subgradient at the "center" of the polyhedral.
  - From the definition of subgradient we have for any w that

$$f(w) \geq f(w^k) + g_k^T(w - w^k),$$

    so any w satisfying $g_k^T(w - w^k) > 0$ will be greater than $f(w^k)$.
  - Adding this constraint corresponds to a plane that "cuts" the polyhedron.

Implementing the Cutting Plane Method

- Many ways to define the "center" of the polyhedron.
  - Goal is to choose a point where all cuts maximally reduce the search space.
- Center of gravity method chooses center of gravity of the set.
  - Requires $O(d \log(1/\epsilon))$ iterations.
  - Difficult to compute exactly.
    - Get same complexity by using average of points sampled from the set.
    - "Hit and run" method is one way to sample over a convex set (not cheap).
- Maximum volume inscribed ellipsoid:
  - Finds largest ellipsoid that is completely contained within the set.
    - Can be solved as a convex optimization.
  - Leads to an $O(d^2 \log(1/\epsilon))$ iteration complexity.

Subgradient Methods: Theory vs. Practice

- Worst-case theoretical rates for convex optimization with subgradients:
  - Best subgradient methods requires $O(1/\epsilon^2)$ iterations.
  - Best cutting plane methods requires $O(d\log(1/\epsilon))$ iterations.

- Various practical methods exist that do not improve worst-case:
  - "Ignore non-smoothness" (use subgradients in smooth optimizer).
  - Bundle methods (incorporate previous subgradients).
  - Min-norm subgradient methods (choose the "best" subgradient).

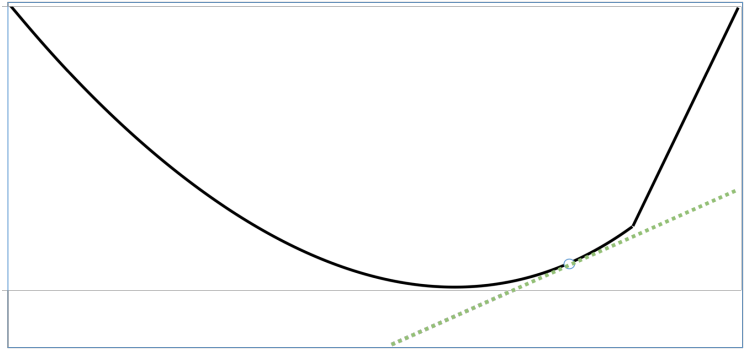
Ignore Non-Smoothness?

- Can we just apply a smooth optimizer to a non-smooth function?
  - Apply accelerated gradient or L-BFGS using subgradients (without smoothing).
- Convex functions are differentiable almost everywhere.
  - So subgradient will typically be a gradient, and line search should give some progress.
- For many problems, this seems to work well.
  - But poorly understood, and can fail.
  - Probably makes the most sense when function is smooth at solution.

Using Multiple Subgradients

- At iteration k, the function value and subgradient give us a lower bound,

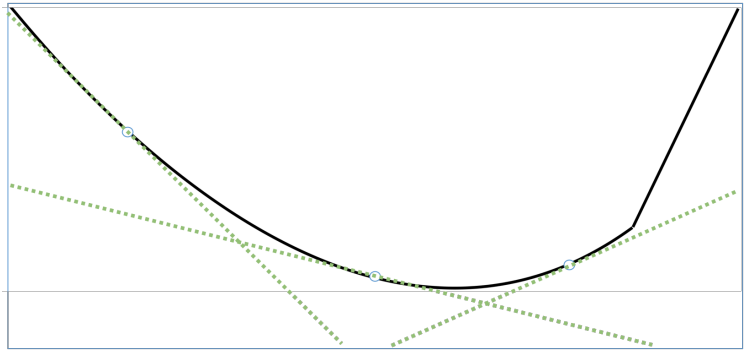
$$f(w) \geq f(w^k) + g_k^T(w - w^k).$$



Using Multiple Subgradients

- We get a tighter bound by using all previous function and subgradient values,

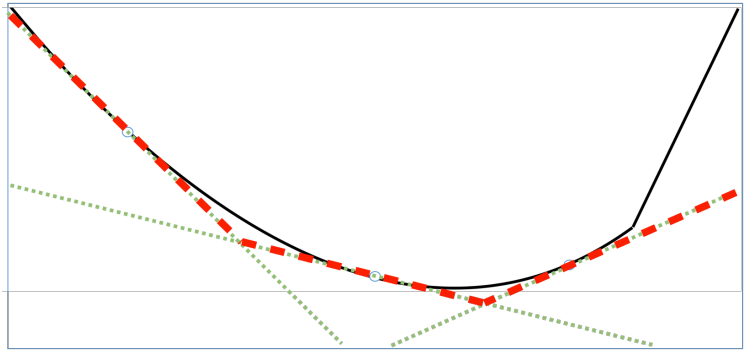
$$f(w) \geq \max_{t \in \{1,\ldots,k\}} \left\{ f(w^t) + g_t^T (w - w^t) \right\}.$$



Using Multiple Subgradients

- We get a tighter bound by using all previous function and subgradient values,

$$f(w) \geq \max_{t \in \{1,\ldots,k\}} \left\{ f(w^t) + g_t^T (w - w^t) \right\}.$$



Bundle Methods

- We can write the subgradient method as using

$$\underset{w}{\text{argmin}} \left\{ f(w^k) + g_k^T(w - w^k) + \frac{1}{2\alpha_k}\|w - w^k\|^2 \right\}.$$

- A common variation on a bundle method uses

$$\underset{w}{\text{argmin}} \left\{ \max_{t \in \{1,\dots,k\}} \left\{ f(w^t) + g_t^T(w - w^t) \right\} + \frac{1}{2\alpha_k}\|w - \bar{w}^k\|^2 \right\},$$

  where on each iteration we set $\bar{w}^{k+1} = w^{k+1}$ or $\bar{w^k} = w^k$ depending on progress.
  - "Serious step" vs. "null step".
    - For ML problems you can replace the squared norm with the regularizer.
  - We may also introduce a second step size as in projected gradient.
- Bundle methods do not improve the worst-case convergence rate.
  - But can converge much faster in practice, though iterations are expensive.
    - Most appropriate when computing subgradients is expensive.

What is the best subgradient?

- We considered the deterministic subgradient method,

$$w^{k+1} = w^k - \alpha_k g_k, \text{ where } g_k \in \partial f(w^k),$$

  under any choice of subgradient.

- Can we instead choose the "best" subgradient?
  - Convex functions have directional derivatives everywhere.
  - Direction $-g_t$ that minimizes directional derivative is minimum-norm subgradient,

$$g_k \in \underset{g \in \partial f(w^k)}{\text{argmin}} \{||g||\}.$$

  - This is the steepest descent direction for non-smooth convex optimization problems.

Minimum-Norm Subgradient Method

- The minimum-norm subgradient method uses

$$w^{k+1} = w^k - \alpha_k g_k, \text{ where } g_k \in \underset{g \in \partial f(w^k)}{\text{argmin}} \{||g||\}.$$

- Some advantages:
  - Solution is a fixed point: $w^* = w^* - \alpha g_*$ since $g_* = 0$.
  - Otherwise, we can satisfy line-search criteria since $-g_k$ is a descent direction.
    - And line searches work with directional derivatives, which exist.

- Some issues:
  - The minimum-norm subgradient may be difficult to find.
  - Convergence not well understood.
    - Not shown to improve worst-case rate over subgradient method.
    - Counter-examples exist where line search causes convergence to sub-optimal values.

Min-Norm Subgradient Method for L1-Regularization

- Consider optimizing a smooth f with (non-smooth) L1-regularization,

$$\operatorname*{argmin}_w f(w) + \lambda \|w\|_1.$$

- The subdifferential with respect to coordinate j has the form

$$\nabla_j f(w) + \lambda \begin{cases} \text{sign}(w_j) & w_j \neq 0 \\ [-1, 1] & w_j = 0 \end{cases}.$$

- The element of the subdifferential with smallest absolute value is given by

$$\begin{aligned} \nabla_j f(w) + \lambda \text{sign}(w_j) \text{ for } w_j \neq 0 \\ \nabla_j f(w) - \lambda \text{sign}(\nabla_j f(w)) \text{ for } w_j = 0, |\nabla_j f(w)| > \lambda \\ 0 \text{ for } w_j = 0, |\nabla_j f(w)| \leq \lambda \end{aligned}$$

- This can be viewed as the steepest descent direction for L1-regularization.
  - Notice that it "keeps variables at 0" if partial derivative at zero is too small.
  - However, the min-norm subgradient does not naturally set variables to 0.

Orthant-Projected Min-Norm Subgradient for L1-regularization

- Min-norm subgrad method with orthant projection for L1-regularization,

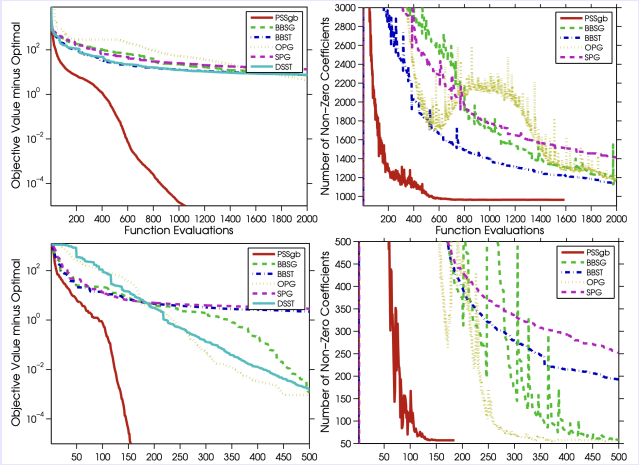
$$w^{k+1} = \text{proj}_{\mathcal{O}(w^k)}[w^k - \alpha_k g_k], \text{ where } g_k \in \underset{g \in \partial f(w^k)}{\text{argmin}} \{||g||\},$$

  where $\text{proj}_{\mathcal{O}(w^k)}[z]$ sets $z_j = 0$ if $\text{sign}(z_j) \neq \text{sign}(w_j)$.
  - So w^{k+1} stays in the same orthant as w^k.

- Has a bunch of appealing properties that make it hard to beat in practice:
  - Orthant-project can set many values to 0 on each iteration.
  - Min-norm subgradient keeps values at 0.
  - Can be combined with line-search (function is smooth over each orthant).
  - Can use clever step sizes like Barzilai Borwein.
  - Can use two-metric projection to implement Newton-like update.
- Convergence and convergence rate properties not well understood at all.
  - Looks like projected gradient with changing set, but individual steps may be bad.

Orthant-Projected Min-Norm Subgradient for L1-regularization

- PSSgb uses min-norm-subgrad/orthantProject/twoMetric/L-BFGS:



(Other methods are first-order, my PhD thesis compares other 1.5-order solvers.)

Summary

- Global optimization of non-convex Lipschitz-continuous functions.
  - Optimal rate is $O(1/\epsilon^d)$, achieved by random search ("harmless" if achieve this rate).
- Subgradients: generalize gradients for non-smooth convex functions.
- Subgradient method: optimal dimension-independent $O(1/\epsilon^2)$ rate for convex f.
  - Does not guarantee decrease in f, but guarantees decrease in distance to solution.
  - Requires similar step sizes to SGD, but Polyak step size allows adaptive steps.
- Projected/stochastic subgradient: same speed as subgradient method.
- Smooth approximations with accelerated gradient give faster rates.
  - But smoothing can destroy structure in solution.
- Cutting plane methods achieve dimension-dependent $O(\log(1/\epsilon))$ for convex f.
  - But iterations are very expensive.
- Practical subgradient methods that improve performance.
  - Ignoring non-smoothness, bundle methods, min-norm subgradient for L1-reg.

- Next time: methods with rate $O(\log(1/\epsilon))$ for specific non-smooth problems.

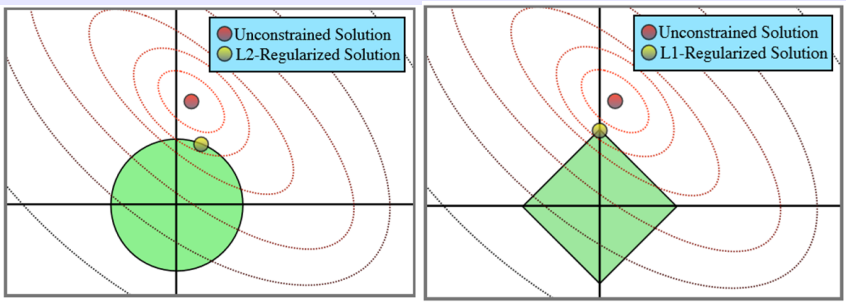
Pruning Global Optimization Algorithms with Lipschitz Constants

- Given the Lipschitz constant L, we can use it to prune parts of the space.
  - From Lipschitz continuity we can get a lower bound on v in terms of any w,
$$f(w) - L\|w - v\| \leq f(v).$$
  - Given the previous iterates $\{w_0, w_1, w_2, \ldots, w^{k-1}\}$, we thus have
$$f(v) \geq \max_{t \in [0,1,\ldots,k-1]} \{f(w_t) - L\|w - v\|\}.$$
  - We can reject any w^k where this bound certifies that
$$f(w^k) \geq \min_{t \in [0,1,\ldots,k-1]} \{f(w_t)\},$$
since such points improve the function value.
- If you know L or have an upper bound on it, this is harmless.
- If you under-estimate L, this is not harmless (could rule out the global minima).
- In practice L is often estimated using $\max_{t,k \mid t \neq k}\{(|f(w_k - f(w_t)|)/\|w_k - w_t\|\}$.
  - This is not harmless.
  - Maximum between this estimate and a sequence satisfying $\Omega((L/\epsilon)^d)$ is harmless.

L1-Regularization vs. L2-Regularization

- Another view on sparsity of L2- vs. L1-regularization using our constraint trick:

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} f(w) + \lambda\|w\|_p \quad \Leftrightarrow \quad \operatorname*{argmin}_{w\in\mathbb{R}^d,\tau\in\mathbb{R}} f(w) + \lambda\tau \text{ with } \tau \geq \|w\|_p.$$



- Notice that L2-regularization has a rotataional invariance.
  - This actually makes it more sensitive to irrelevant features.